

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №1**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Рекурсия**  
**Вариант 9**

Студент гр. 8304

Сани З. Б.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

## **Цель работы.**

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

## **Постановка задачи.**

- 1) проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
- 2) разработать программу, использующую рекурсию;
- 3) сопоставить рекурсивное решение с итеративным решением задачи;
- 4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

9 Разработать программу, которая по заданному простому\_логическому выражению (определение понятия см. в предыдущей задаче), не содержащему вхождений простых идентификаторов, вычисляет значение этого выражения.

## **Описание алгоритма.**

После получения выражения мы разделяем выражение на три категории (первое выражение + знак + второе выражение) в случае ,когда перед логикой стоит скобка ,затем проверяем каждое выражение и затем отрицаем значение выражения, где есть оператор NOT, иначе рекурсивно вычисляем значение этого выражения. Оператор not также рекурсивно определяет, какая логика должна быть отклонена. Затем, после рекурсивного разбиения двух выражений, вы оцениваете оба и возвращаете значение.

### **ALGORITHM:**

```
calculate_expression(input, position)
    if input = "TRUE"
        return true
    if input = "FALSE"
        return false
    if input = "NOT(TRUE)"
        return false
    if input = "NOT(FALSE)"
        return true

    if input[position] = "("
        expr1 = is a substring of input
        expr2 = is a substring of input
        sign = is a substring of input
```

```

if expr1 contains Not statement
    expr1= calculate_expression(!expr1,0)
if expr2 contains Not statement
    expr2= calculate_expression(!expr2,0)
a = calculate_expressionexpr1, 0)
b = calculate_expression(expr2, 0)
return a sign b;

```

### **Спецификация программы.**

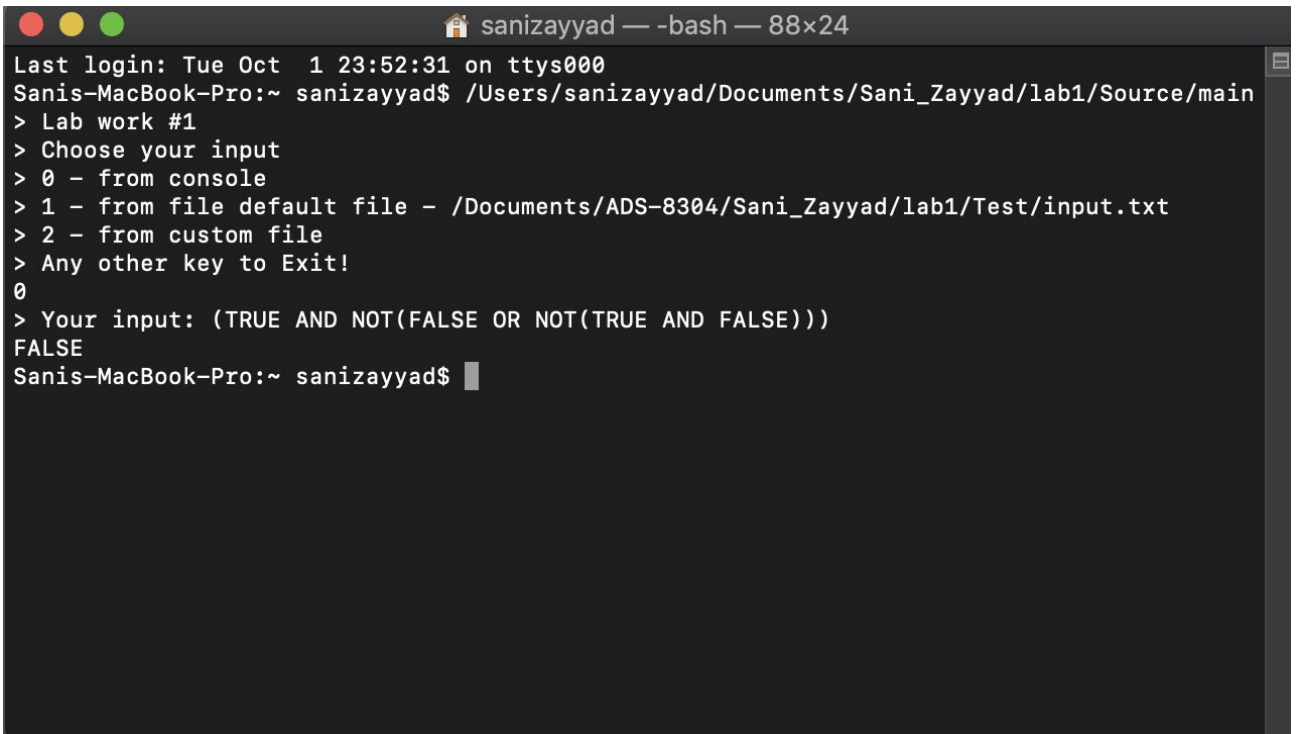
Программа предназначена для синтаксического анализа выражения методом рекурсии.

Программа написана на языке C++. Входными данными является строка из консоли или строка из файла, как указано, но пользователь. Выходными данными являются промежуточные значения проверки выражения который является либо TRUE / FALSE или сообщение об ошибке, когда выражение не является допустимым

### **ПРИЛОЖЕНИЕ.**

#### **1) ТЕСТИРОВАНИЕ:**

**Работа программы для строки(TRUE AND NOT(FALSE OR NOT(TRUE AND FALSE)))**



```

Last login: Tue Oct  1 23:52:31 on ttys000
Sanis-MacBook-Pro:~ sanitayyad$ /Users/sanitayyad/Documents/Sani_Zayyad/lab1/Source/main
> Lab work #1
> Choose your input
> 0 - from console
> 1 - from file default file - /Documents/ADS-8304/Sani_Zayyad/lab1/Test/input.txt
> 2 - from custom file
> Any other key to Exit!
0
> Your input: (TRUE AND NOT(FALSE OR NOT(TRUE AND FALSE)))
FALSE
Sanis-MacBook-Pro:~ sanitayyad$

```

**Таблица результатов ввода/вывода тестирования программы**

Входная строка	Вывод программы
TRUE	TRUE
FALSE	FALSE
NOT(TRUE)	FALSE
NOT(FALSE)	TRUE
(TRUE AND FALSE)	FALSE
(TRUE OR FALSE)	TRUE
(TRUE AND (TRUE OR FALSE))	TRUE
(TRUE AND (TRUE AND FALSE))	FALSE
(TRUE AND (FALSE OR (TRUE AND TRUE)))	TRUE
(NOT(TRUE) OR TRUE)	TRUE
(NOT(TRUE OR FALSE) AND FALSE)	FALSE
(NOT(TRUE AND FALSE) AND NOT(TRUE AND FALSE))	TRUE
(NOT(TRUE OR FALSE) OR NOT(FALSE AND (TRUE AND (FALSE OR TRUE))))	TRUE
(TRUE AND NOT(FALSE OR NOT(TRUE AND FALSE)))	FALSE
)TRUE AND FALSE(	ERROR!!!!

```
sanizayyad — -bash — 94x37
> 0 - from console
> 1 - from file default file - /Documents/ADS-8304/Sani_Zayyad/lab1/Test/input.txt
> 2 - from custom file
> Any other key to Exit!
1
Reading from file:
test #1 "TRUE"
TRUE
test #2 "FALSE"
FALSE
test #3 "NOT(TRUE)"
FALSE
test #4 "NOT(FALSE)"
TRUE
test #5 "(TRUE AND FALSE)"
FALSE
test #6 "(TRUE OR FALSE)"
TRUE
test #7 "(TRUE AND (TRUE OR FALSE))"
TRUE
test #8 "(TRUE AND (TRUE AND FALSE))"
FALSE
test #9 "(TRUE AND (FALSE OR (TRUE AND TRUE)))"
TRUE
test #10 "(NOT(TRUE) OR TRUE)"
TRUE
test #11 "(NOT(TRUE OR FALSE) AND FALSE)"
FALSE
test #12 "(NOT(TRUE AND FALSE) AND NOT(TRUE AND FALSE))"
TRUE
test #13 "(NOT(TRUE OR FALSE) OR NOT(FALSE AND (TRUE AND (FALSE OR TRUE))))"
TRUE
test #14 "(TRUE AND NOT(FALSE OR NOT(TRUE AND FALSE)))"
FALSE
test #15 ")TRUE AND FALSE("
ERROR!!!
Sanis-MacBook-Pro:~ sanizayyad$
```

## Выводы.

За время работы я познакомился с рекурсией, научился работать с этим методом. Я считаю, что метод рекурсии эффективен для этой задачи вычисления значения выражения .

## 2) ИСХОДНЫЙ КОД:

```
#include <iostream>
#include <string>
#include <fstream>

void ProceedInput();
std::string ReadFromConsole();
void ReadFromFile(std::string filename);
bool checkBracket(std::string str);
std::string recursiveNotExpression(std::string str);
bool recursiveExpression(std::string str, int pointer);
void calculate(std::string input);

void ProceedInput()
{
    std::cout << "> Lab work #1" << std::endl;
    std::cout << "> Choose your input" << std::endl;
    std::cout << "> 0 - from console" << std::endl;
```

```

std::cout << "> 1 - from file default file - /Documents/ADS-8304/Sani_Zayyad/lab1/Test/input.txt" << std::endl;
std::cout << "> 2 - from custom file" << std::endl;
std::cout << "> Any other key to Exit!" << std::endl;

int command = 0;
std::cin >> command;
std::cin.ignore();

std::string FilePath = "/Users/sanizayyad/Documents/ADS-8304/Sani_Zayyad/lab1/Test/input.txt";
std::string input;

switch (command)
{
case 0:
    std::cout << "> Your input: ";
    input = ReadFromConsole();
    calculate(input);
    break;
case 1:
    ReadFromFile(FilePath);
    break;
case 2:
    std::cout << "> FilePath: ";
    std::cin >> FilePath;
    ReadFromFile(FilePath);
    break;
default:
    std::cout << "GOODBYE!";
}
}
// this function basically just get input from the console
std::string ReadFromConsole()
{
    std::string input;
    std::getline(std::cin, input);
    return input;
}
//this function iterates the lines of the file and trigger calculate on every line as input
void ReadFromFile(std::string filename)
{
    std::ifstream file(filename);

    if (file.is_open())
    {
        std::cout << "Reading from file:" << "\n";

        int count = 0;

        while (!file.eof())
        {
            count++;
            std::string str;
            getline(file, str);
            std::cout << "test #" << count << " \'" + str + "\'" << "\n";
            calculate(str);
        }
    }
    else
    {
        std::cout << "File not opened";
    }
}
// checking if the brackets are balanced
bool checkBracket(std::string str)
{
    int balanceBracket = 0;
    for (int i = 0; i < str.size(); i++)
    {

```

```

    if(str[0] == ')' || str[str.size()-1] == '{'
        return false;
    }

    if (str[i] == ')')
        balanceBracket--;
    else if (str[i] == '(')
        balanceBracket++;
}
if (balanceBracket == 0)
    return true;
return false;
}
//the main recursive function to check recursiveExpression
bool recursiveExpression(std::string str, int pos)
{
    if (str == "TRUE")
        return true;
    if (str == "FALSE")
        return false;
    if (str == "NOT(TRUE)")
        return false;
    if (str == "NOT(FALSE)")
        return true;

    if (str[pos] == '(')
    {
        pos++;
        bool expression1, expression2;
        std::string string_expression1, string_expression2, sign;

        //I splitted the recursiveExpression three which means for every recursiveExpression there's
        //recursiveExpression1 Sign recursiveExpression2//

        int breakspace = 0;
        int bracket = 0;
        for (breakspace = pos; breakspace < str.size(); breakspace++)
        {
            if (str[breakspace] == ' ')
            {
                if (bracket == 0)
                    break;
            }
            if (str[breakspace] == ')')
                bracket--;
            else if (str[breakspace] == '(')
                bracket++;
        }
        int tmp = breakspace - 1;

        //Expression 1
        string_expression1 = str.substr(pos, tmp);
        //if there's a Not statement in Expression1
        //we call recursiveNotExpression to calculate the NOT(logic)
        if (string_expression1.find("NOT") != std::string::npos)
        {
            string_expression1 = recursiveNotExpression(string_expression1);
        }

        //SIGN
        std::string tmpStr = str.substr(tmp + 2);
        int tmp_2 = tmpStr.find(" ");
        sign = tmpStr.substr(0, tmp_2);

        //Expression 2
        string_expression2 = tmpStr.substr(tmp_2 + 1, tmpStr.size() - tmp_2 - 2);
        if (string_expression2.find("NOT") != std::string::npos)

```

```

    {
        string_expression2 = recursiveNotExpression(string_expression2);
    }

    //recursive on Expression 1
    expression1 = recursiveExpression(string_expression1, 0);
    // recursive on Expression 2
    expression2 = recursiveExpression(string_expression2, 0);

    // the we return the outcome
    if (sign == "AND")
        return (expression1 && expression2);
    else
        return (expression1 || expression2);
    }
}
//this returns the negation of a logic.
std::string recursiveNotExpression(std::string str)
{
    int tmp = str.find("NOT");
    std::string tmp_recursiveExpression = str.substr(tmp + 3, str.size());
    int tmp_closing = tmp_recursiveExpression.find("(");
    tmp_recursiveExpression = tmp_recursiveExpression.substr(0, tmp_closing);
    bool before;

    if (tmp_recursiveExpression.find(" ") != std::string::npos)
    {
        before = recursiveExpression(tmp_recursiveExpression, 0);
    }
    else
    {
        tmp_recursiveExpression = tmp_recursiveExpression.substr(1, tmp_closing - 1);
        before = recursiveExpression(tmp_recursiveExpression, 0);
    }

    std::string notExp;
    if(before == true){
        notExp = "NOT(TRUE)";
    }else{
        notExp = "NOT(FALSE)";
    }

    return notExp;
}
void calculate(std::string input)
{
    if (checkBracket(input) && !input.empty())
    {
        if (recursiveExpression(input, 0) == true)
            std::cout << "TRUE" << std::endl;
        else
            std::cout << "FALSE" << std::endl;
    }
    else
        std::cout << "\t ERROR!!!" << std::endl;
}

int main()
{
    ProceedInput();

    return 0;
}

```